CS 6343: CLOUD COMPUTING
Term Project

Group A1

- TA: Shuai Zhang
- Facility: Shared Cluster in ECSS 4.217
  - 6 machines, each team has its own master, with 5 shared slaves, final demo with all 6 machines
- Project: IaaS cloud middleware
  - Create a cloud environment with a number of servers, allowing users to submit their jobs, scale their jobs
  - Make simple resource management solutions in determining where to place a VM and when to migrate them
- Project steps
  - Cloud environment setup
    - Set up the KVM environment
    - Provide an interface for job submission and scaling
  - Middleware implementation
    - Explore libvirt to achieve VM creation, scaling, and migration
    - Study and summarize papers in VM placement
    - Design and develop a VM placement and migration decision algorithm
  - System monitoring
    - Install one of the cloud monitoring system to observe various behaviors of the system
      - Continuously display memory load, CPU load, IO load, etc.
      - For individual VMs, individual nodes, etc.
      - Create the interface for the monitoring result display
  - Obtain workloads from A2, and perform cloud middleware management
  - Analyze the results
- Papers
  - D. Ardagna, M. Trubian, and L. Zhang, "SLA based resource allocation policies in autonomic environments," Journal of Parallel and Distributed Computing, vol. 67, pp. 259-270, 2007.
  - J. Almeida, V. Almeida, D. Ardagna, Í. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," Journal of Parallel and Distributed Computing, vol. 70, pp. 344-362, 2010.
  - On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach
  - On Resource Management for Cloud Users: A Generalized Kelly Mechanism Approach
  - SLA-aware virtual resource management for cloud infrastructures
  - VMware Distributed Resource Management: Design, Implementation, and Lessons Learned
  - Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds

Group A2

- TA: Shuai Zhang
- Facility: Shared Cluster in ECSS 4.217
  - 6 machines, each team has its own master, with 2 shared slaves
- Project: Cloud benchmark workload
  - Create and install a suite of cloud benchmark program as input jobs to the cloud middleware
- Project steps

- Create and install a suite of cloud benchmark program
    - TPC-W: http://www.tpc.org/tpcw/
    - YCSB:
        - http://labs.yahoo.com/news/yahoo-cloud-serving-benchmark/
        - https://github.com/brianfrankcooper/YCSB/wiki
    - HPL: http://www.netlib.org/benchmark/hpl/
    - Hadoop benchmarks: TestDFSIO, TeraSort, nnbench, mrbench
        - http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/
    - CloudSuite: http://parsa.epfl.ch/cloudsuite/cloudsuite.html
    - …
- Explore VM creation and create VMs for the benchmark programs
- Run the VMs in different cloud platforms
    - A1's cloud environment, Amazon cloud, etc.
- Adjust the parameters of the benchmark programs to create different workloads
- Create simulated workloads and submit them to A1's cloud environment
- Under similar settings, compare the behavior of A1's cloud and Amazon cloud

---

Group B

- TA: Yongtao Huang
- Facility: Shared Cluster in ECSS 4.217
    - 5 machines, each team has its own master, with 3 shared slaves
- Project: Cloud file systems
    - Install a few famous cloud file systems, explore their features and compare their performance
- Project steps
    - Explore different file systems
        - HDFS http://hadoop.apache.org/
        - Swift: http://swift.openstack.org/
        - Ceph: http://ceph.com/
    - Install the above file systems on a cluster
    - Identify a feature vector (what features should be considered if a user needs to select a file system to use)
        - Look up time
        - Access latency, access throughput, directory service latency, etc.
        - Load balancing features and performance
        - Consistency model and solutions, availability solutions, etc.
        - …
        - Other special features that are unique to a certain file system
    - Use IOzone to create the simulated file system and create file system
    - Evaluate the file systems based on the feature vector
        - http://www.iozone.org/

Group C

- TA: Yongtao Huang
- Facility: Shared Cluster in ECSS 4.217
    - 4 machines, each team has its own master, with 2 shared slaves, final demo with all 4 machines
- Project: Directory structure maintenance
    - Compare different methods in implementing directory files
        - In Unix system, each directory is a file by itself

        - In many distributed file system, files are treated as individual elements during placement among the distributed servers. Directory structure is handled separately.
            - Solution 1: Use a centralized server to store the entire directory
            - Solution 2: Treat directory files as regular files
            - Solution 3: Ceph solution
            - Solution 4: Merge a subtree of directories into one file, with a fix number of levels
- Project steps
    - Develop the four directory structure maintenance solutions and compare their performance
    - Consider directory creation, deletion, rename, cd, ls operations
    - Generate a huge directory structure
    - Develop a directory request generation program to generate requests and evaluate the performance of different solutions

---

Group D

- TA: Yongtao Huang
- Facility: Shared Cluster in ECSS 4.213
    - 6 machines, each team has its own master, with 4 shared slaves, final experimental study should be done with 8 machines (2 additional will be added later)
- Project: Load balancing in DHT based file systems
    - Develop a load balancing solution for Swift like file systems
- Project steps
    - Study the load balancing solutions in DHT based file systems
        - Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems (05)
        - Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks (07)
        - The server reassignment problem for load balancing in structured p2p systems (08)
    - Variation of DHT
        - Sloppy hashing and self-organizing clusters
        - A self-organizing flock of condors
        - Self-organization in spontaneous networks: the approach of DHT-based routing protocols
    - Implement a load balancing solution on a Swift-like file system
    - Compare the performance with other DHT-based load balancing solutions
- Papers
    - J. Byers, J. Considine. “Simple Load Balancing for Distributed Hash Tables”. Peer-to-Peer System II. Pages: 80-87, 2003.
    - D. Liu, F. Chen, G. Yin. “LSB-Chord Load Balancing in DHT based P2P systems under Churn”. In Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010). Pages: 466-470, 2010.
    - Y. Chang, H. Chen, S. Li. “A Dynamic Hashing Approach to Supporting Load Balance in P2P Systems”. In Proc. 28th International Conference on Distributed Computing Systems Workshops (ICDCS ’08). Pages: 429-434, June 2008.
    - C. Chen, K. Tsai. “The server reassignment problem for load balancing in structured p2p systems”. IEEE Transactions on Parallel and Distributed Systems. Pages: 234-246, 2008.

Group E

- TA: Yongtao Huang
- Facility: Shared Cluster in ECSS 4.217
    - 4 machines, each team has its own master, with 2 shared slaves, final demo with all 4 machines

- Project: Ceph file system
  - Develop a simple version of the Ceph file system, focusing on its naming service solution
- Project steps
  - Study the Ceph file system solutions
  - Develop the basic data placement algorithm of Ceph
  - Add the Ceph solution for fault tolerance and load balancing
    - When there is a node failure or a node join, how the Ceph placement algorithm handles the failure
    - When the system performs load balancing, how will the Ceph placement algorithm changes its directory management solution
  - Compare the Ceph solution with other solutions
    - Central directory solution in HDFS and DHT solution in Swift

---

Project submissions

- Regular meetings
  - There will be weekly meetings with the TA and bi-weekly meetings with the Professor
  - Schedule and milestones will be defined during the meetings
- Mid-term submissions
  - Your working code
    - Though your code is not for the entire project, it should implement some functions required in the project and should be fully working
  - Midterm project report (contents is the same as final report)
  - **Midterm report due: Oct 23, 2015**
  - **Midterm demo: Subsequent group meeting time**
- Final submissions
  - Your working code
  - Final project report
  - **Final report due: Dec 7, 2015**
    - Final report will be progressively revised from midterm report (revised report should be submitted till it is satisfactory)
  - **Final demo** will be schedule during midterm demo
    - Teams that fall behind will have meetings every week to ensure the progress is caught up
- Submission guideline
  - Each team only needs one submission, team leader should make the submission through e-learning
  - <team-label>: A11, A12, A21, A22, B1, B2, C1, C2, D1, D2, E1, E2
  - During submission, attach the doc file and the file name has to be <team-label>.doc or <team-label>.docx (e.g., A11.doc)
    - Do not submit pdf
  - For teams that mainly do coding (C, D, E)
    - Zip or tar your code, file name has to be <team-label>.zip
    - Attach the zip file during e-learning submission
  - For teams with a large number of VMs (A1, A2, B)
    - Zip or tar your code, file name has to be <team-label>.zip
    - Prepare all the VMs that are needed to test your system
    - Upload the zip file and the VMs to MS OneDrive, provide the url in your report
- Required information in the report (can include more than listed)
  - In the cover page, provide team-label, title, and team members
  - Introduction
    - Goal of the project

    - What you would offer in your system and why what you offered is important
- Study of related work
  - Summary of related works (in paper or similar products available)
  - How your project is different from or is similar to some existing works
- Approach
  - System architecture
    - Activity diagram (workflow)
    - Architecture diagrams (from high level to low level decomposition of the system)
    - Description of the nodes in the architecture
  - Detailed design
    - Including the APIs of each important component, the algorithms used in some components, etc.
  - Implementation details
    - List the code files and VMs that are in your system
    - Provide a structure of the code files and describe each code file and each VM in the list
      - The relations of the code files and what they are doing should have been in the detailed design section, you can relate the code files to those in the design section
  - Problems encountered and how they are resolved
    - This is important, it is the basis for us to decide the efforts you have actually put in the project
  - ...
- Experimental results
  - Clearly state the control parameters and the metrics for measurement
  - Experimentation needs to be thorough and results need to be easy to read (e.g., use graphs and table)
  - Some teams need to coordinate with each other in experimentation
    - A1 and A2
    - D and E and TA
- Installation manual, including existing open source used (no need to give installation guide for open sources, just their urls) and how to setup your programs
- User manual, discussing how to use your system
- Workload distribution
  - Include a high level summary and a detailed list
  - The detailed list include tasks done by each member in the team week by week (just attach the log you prepared every week)